컴퓨터 비젼 기말과제: Edge detector 구현에 대한 결과보고서

2017103580 사회학과 김정운

1.기존 알고리즘의 문제 및 한계 분석

edge를 찾는 과정은 크게 두가지로 나뉘는데, 먼저 이미지의 잡음을 줄인 다음 주변 픽셀과의 밝기 차이를 구한다. 그리고 그 값이 일정한 기준(threshold)보다 크면 해당 픽셀을 edge로 선언한다. 구체적으로 이미지의 잡음을 제거하기 위해 box-filter와 Gaussian-filter를 사용하는데, 두 방법 모두 특정 모형을 가정하고 있다. 즉, 두 filter가 유연하지 않음을 의미하고 이로 인해 처리해야 할 이미지의 color나 밝기가 급격하게 변하면, 좋은 성능을 내기 쉽지 않을 수 있다. 또한 두 필터에서 적절한 사이즈(또는 분산)를(을) 설정하는 것이 쉽지 않다. 특히 후자와 관련된 문제는 filter에서 사용할 샘플크기(또는 분산)이 너무 크면 모수가 다른 픽셀들도 연산에 포함되기 때문에 편차가 발생하고, 반대로 작으면 중심극한정리를 만족시킬 수 없다는 것이다. 이러한 전처리(특히 box-filter)는 데이터를 왜곡시킴으로써 edge가 두꺼워지는데, 이를 해결하는데 또 다른 연산(canny edge detection 등)을 해야 한다는 단점도 있다.

또한 Gaussian edge detector는 각 픽셀마다 지수함수에 대한 연산을 수행하는데, 매우 큰 M에 대해 을 연산을 각 픽셀마다 해야 하며, 이는 알고리즘의 연산 복잡도를 높인다. 그리고 기존 알고리즘들이 밝기의 변화량으로 edge를 판단하는데, 밝기의 표현력은 한계가 있다. 즉, 이는 색깔이 서로 다름에도 불구하고 RGB 값의 합이 같으면 밝기 차이가 적지 때문에 edge로 분류되지 않을 수 있다는 것이다. 마지막으로 threshold를 설정하는데 주관적일 수 있다는 문제를 가지고 있다.

그래서 이번 레포트에서는 위의 문제를 해결할 수 있는 edge detector 알고리즘을 제시하고자 한다. 간단하게 설명하면, 우선 edge를 판단할 때 사용할 정보는 기존의 밝기 뿐 만 아니라 RGB 또는 밝기+RGB도 된다. 또한 각 픽셀을 특정 그룹으로 분류한 다음, 픽셀의 주변(위아래 또는 좌우)를 참조한다. 만약 해당 픽셀과 주변의 집단이 서로 다르면 그 픽셀을 edge로 분류하며, 사용자가 원하면 해당 픽셀이 노이즈 여부도 확인하여 이를 edge 분류에 반영한다. 이러한 분류는 군집 알고리즘처럼 모든 픽셀에 대해 이뤄지기 때문에 사용하는 정보가 많을 뿐만 아니라, local 영역에서만 값을 비교하지 않기 때문에 유연성도 기존보다 높다고 할 수 있다.

2.제안한 알고리즘 설명

아래 <참조1>에서는 이미지 전처리 방법에 대한 의사코드를 보여주고 있으며, <참조2>는 특정 픽셀의 edge 여부를 결정하는 알고리즘이다. 이미지 전처리는 유사도를 묶는 방법에 따라 3가지로 나뉠 수 있지만, 공통적으로 특정 픽셀의 값이 결정적이거나 확률변수여도 분산이 매우 작다는 것을 전재로 하고 있다. 이는 픽셀들을 하나의 그룹으로 분류할 때 다른 군집 알고리즘처럼 집단의 평균을 사용하지 않고, 그룹의 특정 픽셀을 분류할 객체와 비교하여 유사도를 판단하는 것과 관련이 있다. 만약 픽셀들의 분산이 크면 이러한 방식은 픽셀의 대표성에 문제를 제기하여 연산의 신뢰성을 떨어트린다. 따라서 해당 알고리즘을 사용하기 위해서는 특정 픽셀의 값이 일정해야 하며, 이를 확인하기 위해 같은 사진을 여러 번 얻어서 RGB 값을 비교할 수 있다.

이미지 전처리에서 사용할 수 있는 정보는 밝기, 색, 밝기+색이다. 우선 밝기는 기존 알고리즘과 마찬가지로 RGB의 평균값을 사용하지만, 유사도를 측정할 때 유클리드 거리를 활용한다. 특정 그룹 내 픽셀의 밝기와 다른 픽셀의 차이가 임계치보다 작으면 후자를 해당 집단으로 분류하고, 그렇지 않으면 객체를 새로운 그룹으로 분류한다. 그리고 그 생성된 그룹의 대표 값을 최초로 분류된 픽셀의 정보다. 또한 최적의 edge를 설정하는데 임계치는 이미지마다 다른데 이를 통계적으로 결정하기 위해 각 픽셀의 밝기와 평균값의 차이를 구한다. 그리고 그 값의 평균으로부터 유의미하게 떨어진 값을 threshold로 선정한다는 것이다. 이는 기존 통계적 검정과 동일한 방법으로 threshold를 결정하는 것으로, 밝기 차이 분포 하에서 특정 픽셀과 다른 밝기의 차이가 유의미하게 크면 다른 그룹으로 분류할 수 있다.

이를 수행하는 방법은 다음과 같다. 각 픽셀과 전체 이미지의 평균 간의 차이(절대값)를 구하고 이를 오름차순으로 정렬한다. 그리고 유의수준이 5%인 p-value 2개와 중앙값을 임계치(threshold)로 취할 것이다. 이처럼 threshold를 데이터로부터 결정할 수 있기 때문에 사용자의 주관성을 배제할 수 있다. 그리고 임계치가 크면 해당 픽셀을 특정 그룹으로 묶을 가능성이 높으며, 이는 box-filter의 사이즈를 키우는 것과(또는 Gaussian의 분산을 키우는 것) 같다. 밝기 차이의 평균으로부터 유의미하게 큰 값은 유의미한 edge를 얻을 때 사용할 수 있으며, 차이가 평균으로부터 작은 것은 threshold가 작기 때문에 잡음을 판단하는데 활용할 수 있다. 중앙값을 제공하는 이유도 차후 threshold를 여러 개 사용하여 edges를 얻을 수 있는 것을 지원하고자 하기 때문이다.

색으로 특정 픽셀을 그룹으로 분류하는 방법은 RGB 공간에서의 내적을 이용한다. 만약 RGB 공간에서 두 객체의 color 벡터가 서로 같은 방향에 있으면 내적은 1이 되고, 이는 두 객체의 색깔이 이 유사하다는 것을 의미한다. 또한 RGB 공간은 항상 양수만을 취할 수 있기 때문에 내적의 최소값은 0이 된다. 그래서 특정 그룹 대표 벡터와 픽셀의 RGB간의 내적을 구하고 이것이 임계치보다 크면, 해당 픽셀을 그 집단으로 분류한다. 그렇지 않으면 객체를 새로운 그룹으로 분류하고, 그 값을 생성된 그룹의 대표 값으로 선언한다. 마찬가지로 색깔과 밝기를 동시에 사용하는 방법은 기존 RGB공간에 밝기를 추가한 4차원 공간(RGB+밝기)에서 객체와 그룹의 내적을 구하고, 이를 임계치와 비교하는 것이다. 이러한 내적연산은 행렬로 표현될 수 있기 때문에 그래픽카드에서 연산하여 속도를 높일 수 있다.

<참조1: 이미지를 전처리 하는 방법에 대한 의사코드>

|  |
| --- |
| def pre\_processing(Image, threshold):  result =deepcopy(Image) #2차원 리스트이며, 그 크기는 입력 받은 이미지의 것과 같다.  group=[image[0][0]] #픽셀들을 그룹으로 분류하기 위해 초기값을 설정해야 한다.  result[0][0]=[group[0], 0] #픽셀의 정보(RGB 또는 밝기)와 픽셀의 그룹 index를 저장한다.  while(이미지의 모든 픽셀을 분류했는가):  similarity=min(group-Image[i][j]) #객체와 유사도가 가장 높은 그룹을 찾고 유사도를 저장한다.  group\_index=argmin(group-Image[i][j]) #유사도가 가장 높은 그룹의 인덱스를 저장한다.  if(similarity>threshold): #유사도가 임계치보다 높으면 픽셀을 해당 그룹으로 분류한다.  result[i][j]=[Image[i][j], group\_index]  else: #유사도가 임계치보다 낮으면 새로운 그룹으로 픽셀을 분류하고 그 값 대표로 취한다.  group.add(image[i][j]); new\_group\_index=새로운 그룹으로 분류된 픽셀의 그룹 index  result[i][j]=[Image[i][j], new\_group\_index]  return result |

<참조2: 픽셀의 edge 여부를 결정하는 방법에 대한 의사코드>

|  |
| --- |
| def get\_edges(pre\_prcessing(Image, threshold):  result=deepcopy(Image) #반환할 값으로 크기는 Image와 같다.    while(이미지의 모든 픽셀에 대한 edge여부를 확인했는가):]  #특정 픽셀과 좌우 또는 상하의 픽셀간 그룹이 다르면 해당 픽셀을 edge로 분류한다  edges\_test1=(result[i][j][1]!= result[i][j+1][1]) or result[i][j][1]!= result[i][j-1][1])) #좌우비교  edges\_test2=(result[i][j][1]!= result[i-1][j][1]) or result[i][j][1]!= result[i+1][j][1])) #상하비교  if(edges\_test1 or edges\_test2):  result[i][j]=1  else:  result[i][j]=0  return result |

이처럼 각 픽셀을 특정 그룹으로 분류하는 것은 box 또는 gaussian filter처럼 데이터의 잡음을 줄이는 과정이다. 왜냐하면 그룹으로 분류한다는 것은 해당 픽셀의 RGB 또는 밝기를 집단의 값으로 대체하는 것이기 때문이다. 이러한 방법은 box와 gaussian처럼 픽셀들의 평균을 구하지 않기 때문에 중심극한정리를 사용할 수 없지만, 모수를 모르는 상황에서 분산의 크기가 크지 않다고 가정했기 때문에 평균과 특정 객체 간의 차이는 크지 않다. 따라서 평균 대신 특정 객체 값을 사용해도 무방하며, 오히려 평균을 계산하지 않아 연산속도를 높일 수 있다. 이러한 전치리 과정은 군집 알고리즘에서 영감을 받았기 때문에 다른 clustering 방법론을 적용할 수 있다.

제안한 알고리즘의 big-o를 구해보면, mxn 이미지가 있고 덧셈을 기준으로 삼을 때 mng가 된다(g는 max(그룹의 개수)). 같은 이미지에 대해 크기가 kxk인 box-filter의 big-o는 mn\*(k^2)이고 gaussian은 mnE가 된다(E는 의 E이며 이는 보통 매우 큰 수다). 시뮬레이션에서 제안한 알고리즘의 군집 개수가 k^2보다 큰 경우가 많기 때문에 box-filter보다 연산량은 많지만, gaussian보다는 연산 횟수가 적다고 할 수 있다. 하지만 기존 방법이 해당 픽셀의 주변 정보를 사용하기 때문에 local property를 갖는 반면, 제안한 전처리 알고리즘은 모든 픽셀에 대해 비교를 수행하기 때문에 global property를 갖는다. 이로 인해 기존 알고리즘이 픽셀 주변 값을 계산하여 생긴 편차를 막을 수 있으며, RGB 또는 밝기가 유사한 픽셀들을 같은 그룹으로 분류할 수 있다. 즉, 유연성이 높은 동시에 정보 손실을 줄일 수 있다는 것이다.

이러한 전처리를 바탕으로 해당 픽셀을 edges로 분류하는 방법은 먼저 특정 픽셀과 좌우(또는 상하)에 있는 픽셀의 그룹을 비교한다. 여기서 전자와 다른 그룹으로 분류된 픽셀이 있으면, 해당 픽셀을 edge로 분류한다. 이는 잡음에 취약하기 때문에 필요한 경우 다음과 같은 방법을 기존 알고리즘에 추가한 것을 사용할 수 있다. 픽셀의 neighborhood를 참조하여 각 픽셀의 그룹을 확인한 다음, 조사한 픽셀 개수 대비 그룹의 종류를 비(Ratio)로 구한다. 만약 그 값이 임계치보다 크면 해당 픽셀을 edge로 분류하지 않는다. 이는 기존 알고리즘에서 잡음을 판단할 때 주변 픽셀들의 orientation을 확인하는 것과 비슷하다. 즉, 특정 픽셀의 neighborhood가 smooth하지 않기 때문에 픽셀의 주변이 서로 다른 값을 취함으로써 다양한 종류의 그룹이 있지만, 그 픽셀이 이미지에서 객체의 경계를 구성하지 않기 때문에 해당 픽셀을 edge로 분류하지 않는다. 이는 box나 gaussian filter의 smoothing과 비슷한 역할을 하지만, 특정 픽셀의 정보를 주변 객체들의 연산결과로 대체하지 않기 때문에 편차는 없다.

3.시뮬레이션(Simulation)

우선 필자는 위에서 설명한 알고리즘을 lubuntu(우분투의 파생 OS)에서 Jupyter notebook을 통해 python3.8.10으로 작성했다. 또한 시뮬레이션 한 컴퓨터의 주요 스펙은 CPU(i3-9100), Ram 16GB, SSD 120GB이다. <그림1> 을 보면 알고리즘의 성능을 확인하기 위해 사용된 사진은 총 2장으로, 첫 번째는 기말과제로 제공된 사진(이하 샘플2)이다. 이를 선정한 이유는 광자의 특성상 각 픽셀에 저장되는 정보가 확률변수이기 때문이다. 즉, 픽셀 주변에 잡음이 존재한 상황에서 해당 알고리즘이 edges를 잘 선정할 수 있는가를 확인하고자 한다. 두 번째 사진은 게임 스타크래프트2 플레이 장면(이하 스타2)으로, 이는 객체와 배경의 밝기 및 RGB가 유사하다는 특징을 가지고 있다. 즉, edges를 잘 뽑아 내기 어려운 사진으로, 이러한 상황에서 edges를 잘 뽑아낼 수 있는 가는 중요한 문제라고 판단했기 때문에 해당 사진을 테스트에 사용했다.

시뮬레이션 결과를 요약하면 다음과 같다. 우선 밝기의 threshold가 낮을수록 또는 RGB(RGBL: 기존 RGB공간에 밝기 정보를 추가한 4차원 공간)에서 threshold(유사도)가 높을수록 잡음이 많고 디테일이 높아진다. 이는 값이 조금이라도 다르면 다른 그룹으로 분류하기 발생한다. 그리고 잡음을 제거하고 edge를 얻는 알고리즘이 노이즈가 많은 상황(밝기의 threshold가 낮거나 RGB(또는 RGBL)의 유사도가 높음)에서 효과적이다. 이러한 잡음 제거는 특히 RGB 또는 RGBL에서 임계치를 높게 설정해도 좋은 edge를 얻을 수 있으며, 이는 심지어 임계치가 낮을 때의 edge보다 좋기도 하다. 그러나 임계치가 높아서 잡음이 제거된 상황에서는 edge detect 알고리즘 간 차이가 발생하지 않는다. 또한 밝기가 유사한 상황에서 color 정보(또는 color와 밝기)를 사용한 데이터 전처리가 밝기 만을 이용할 때보다 좋은 edge를 얻지만, 불연속적으로 edges 얻거나 주변 픽셀들 간 밝기가 유사하면 edge로 분류하지 못할 수 있다.

우선 <그림2-1>과 <그림2-2>에서 밝기를 활용하여 샘플2에 대한 전처리 및 edge 감지 결과를 보면, threshold와 디테일이 반비례하고 잡음 제거는 비례한다는 것을 알 수 있다. 하지만 threshold가 median 일 때와 유의수준 0.95일 때의 결과가 비슷하다는 것을 알 수 있는데, <그림5-2>의 다른 데이터에 대한 결과를 보면 샘플2에서만 그러한 상황이 발생한다는 것을 알 수 있다. 그리고 threshold가 median이상인 경우 디테일이 매우 떨어지는데, 그림1의 샘플2 사진을 보면 해당 사진의 밝기가 유사한 것과 관련이 있다고 추측할 수 있다. 즉, 밝기가 비슷해서 이를 기반으로 한 edge 추출의 성능이 나쁘다는 것이다. 그리고 edge를 결정할 때 노이즈 제거 효과는 잡음이 많은 상황(thres hold가 005(그림2-1 참조)))에서만 효과적이다.

<그림3-1>과 <그림3-2>를 통해 color를 통한 샘플2 전처리 및 edge를 얻은 결과를 살펴보면, 우선 전반적으로 밝기를 이용했을 때보다 디테일이 높다. 하지만 밝기를 이용했을 때 edge로 선언된 픽셀이 color를 이용할 때는 edge로 선언되지 않았다. 이는 픽셀 주변의 하늘 색과 지붕 색이 유사함으로써 발생했다고 추측한다. 여기서 주목할 것은 일부 영역에서 <그림3-2>의 edge(0.9 5)에 노이즈를 제거한 결과가 edge(0.9)에 잡음을 제거한 것보다 성능이 좋다는 것이다. 구체적으로 전자가 후자보다 <그림1>의 샘플2에서 노란색이 있는 영역을 잡음없이 잘 표현하고 있다.

<그림4-1>과 <그림4-2>에서 color와 밝기를 동시에 사용한 결과이다. 이를 보면 threshold가 0.95이고 잡음을 제거했다고 할 때, 일부영역에 대해서는 컬러 만을 사용하여 얻은 edge보다 컬러와 밝기를 동시에 사용한 edge가 디테일이 높고 잡음도 적다. 하지만 노란색이 있는 부분은 전자(컬러만 사용)가 후자(컬러와 밝기 사용)보다 잡음이 적고 디테일이 높다. 이러한 차이가 발생한 이유에 대해서는 현재 해석하지 못하고 있으며, 이에 대해 연구해야 한다.

<그림 5-1>과 <그림5-2>는 밝기를 활용하여 스타2 이미지를 전처리하고 edge를 얻었다. 이를 <6-2>와 <7-2> 비교했을 때 성능이 매우 안 좋다는 것을 알 수 있다. 데이터의 특성 상 객체와 배경이 유사한 곳이 많기 때문에 밝기를 이용하면 좋은 edge를 얻을 수 없다. 그리고 <6-2>에서는 threshold에 상관없이 잡음을 제거한 edge detector가 그렇지 않은 것보다 좋은 edge를 얻었다는 것을 알 수 있다. 구체적으로 잡음을 제거하지 않았을 때는 객체 내부의 일부 픽셀들이 edge로 분류되었는데, 잡음을 제거하면 그러한 픽셀들은 더 이상 edge가 아니다. <그림 7-2>를 보면 edge를 분류할 때 threshold를 높게 설정해도 잡음을 제거하면 threshold가 낮을 때보다 좋은 edge를 얻을 수 있다는 것을 알 수 있다. 특히 <그림 7-2>에서 잡음을 제거하여 얻은 edge는 그렇지 않을 때보다 객체 안쪽 영역에 있는 픽셀을 edge로 잘못 분류하는 경우가 적다.

정리하면 어떤 전처리 방법과 edge detect 알고리즘이 다른 방법보다 항상 우월하지 않으며, 데이터의 특성(밝기나 color의 유사성 등)에 따라 알고리즘 성능이 달라진다. 그래서 어떤 전처리 방법을 사용하고 edge를 감지해야 하는가를 결정하기 위한 한 가지 방법은, 우선 밝기와 color의 분포를 살펴봄으로써 이미지의 특성을 파악하는 것이다. 예를 들어 밝기의 분포가 균일분포로 수렴한다면. 해당 이미지는 밝기로 edge를 얻는 것은 안 좋은 결과를 얻을 수 있다고 판단할 수 있다. 따라서 색깔 또는 색깔과 밝기를 동시에 이용하여 데이터 전처리를 함으로써 edge를 얻는 것이 효과적일 수 있다.

<그림1: 샘플2와 스타크래프트2 플레이 사진>

|  |  |
| --- | --- |
|  |  |

<그림2-1: 샘플2에 대한 이미지 전처리(밝기) 결과>

괄호 안에 있는 것은 threshold이며 095는 밝기 값의 차이에서 평균보다 큰 값이 해당 분포에서 나올 확률이 5%인 유의수준이다. 005는 반대로 평균보다 작은 값이 해당 분포에서 나올 확률이 5%인 유의수준이다. median은 밝기 값 차이의 중앙값이다.

|  |  |  |
| --- | --- | --- |
| 샘플2 밝기 전처리(005) | 샘플2 밝기 전처리(median) | 샘플2 밝기 전처리(095) |
|  |  |  |

<그림2-2: 샘플2에서 밝기를 통해 얻은 edge 결과>

|  |  |  |
| --- | --- | --- |
| 샘플2 밝기 edge(005) | 샘플2 밝기 edge(median) | 샘플2 밝기 edge(095) |
|  |  |  |
| 샘플2 밝기 edge(005): 잡음제거함 | 샘플2 밝기 edge(median) 잡음제거함 | 샘플2 밝기 edge(095) 잡음제거함 |
|  |  |  |

<그림3-1: 샘플2에서 color를 통해 얻은 전처리 결과>

|  |  |  |
| --- | --- | --- |
| 샘플2 색깔 전처리(유사도 0.8) | 샘플2 색깔 전처리(유사도 0.9) | 샘플2 색깔 전처리(유사도 0.95) |
|  |  |  |

<그림3-2: 샘플2에서 color를 통해 얻은 edge 결과>

|  |  |  |
| --- | --- | --- |
| 샘플2 색깔 edges(유사도 0.8) | 샘플2 색깔 edges(유사도 0.9) | 샘플2 색깔 edges(유사도 0.95) |
|  |  |  |
| 샘플2 색깔 edges(유사도 0.8), 잡음 제거함 | 샘플2 색깔 edges(유사도 0.9), 잡음제거함 | 샘플2 색깔 edges(유사도 0.95), 잡음제거함 |
|  |  |  |

<그림4-1: 샘플2에서 color와 밝기를 통해 얻은 전처리 결과>

|  |  |
| --- | --- |
| 샘플2 색깔과 밝기 전처리(유사도 0.9) | 샘플2 색깔과 밝기 전처리(유사도 0.95) |
|  |  |

<그림3-2: 샘플2에서 color와 밝기를 통해 얻은 edge 결과>

|  |  |
| --- | --- |
| 샘플2 색깔과 밝기 edges(유사도 0.9) | 샘플2 색깔과 밝기 edges(유사도 0.95) |
|  |  |
| 샘플2 색깔과 밝기 edges(유사도 0.9), 잡음 제거함 | 샘플2 색깔과 밝기 edges(유사도 0.95), 잡음제거함 |
|  |  |

<그림5-1: 스타2에 대한 이미지 전처리(밝기) 결과>

|  |  |  |
| --- | --- | --- |
| 스타2 밝기 전처리(005) | 스타2 밝기 전처리(median) | 스타2 밝기 전처리(095) |
|  |  |  |

<그림5-2: 스타2에서 밝기를 통해 얻은 edge 결과>

|  |  |  |
| --- | --- | --- |
| 스타2 밝기 edge(005) | 스타2 밝기 edge(median) | 스타2 밝기 edge(095) |
|  |  |  |
| 스타2 밝기 edge(005): 잡음제거함 | 스타2 밝기 edge(median) 잡음제거함 | 스타2 밝기 edge(095) 잡음제거함 |
|  |  |  |

<그림6-1: 스타2에서 color를 통해 얻은 전처리 결과>

|  |  |  |
| --- | --- | --- |
| 스타2 색깔 전처리(유사도 0.8) | 스타2 색깔 전처리(유사도 0.9) | 스타2 색깔 전처리(유사도 0.95) |
|  |  |  |

<그림3-2: 샘플2에서 color를 통해 얻은 edge 결과>

|  |  |  |
| --- | --- | --- |
| 스타2 색깔 edges(유사도 0.8) | 스타2 색깔 edges(유사도 0.9) | 스타2 색깔 edges(유사도 0.95) |
|  |  |  |
| 스타2 색깔 edges(유사도 0.8), 잡음 제거함 | 스타2 색깔 edges(유사도 0.9), 잡음 제거 함 | 스타2 색깔 edges(유사도 0.95), 잡음제거 함 |
|  |  |  |

<그림7-1: 스타2에서 color와 밝기를 통해 얻은 전처리 결과>

|  |  |
| --- | --- |
| 스타2 색깔과 밝기 전처리(유사도 0.9) | 스타2 색깔과 밝기 전처리(유사도 0.95) |
|  |  |

<그림7-2: 샘플2에서 color와 밝기를 통해 얻은 edge 결과>

|  |  |
| --- | --- |
| 스타2 색깔과 밝기 edges(유사도 0.9) | 스타2 색깔과 밝기 edges(유사도 0.95) |
|  |  |
| 스타2 색깔과 밝기 edges(유사도 0.9), 잡음 제거함 | 스타2 색깔과 밝기 edges(유사도 0.95), 잡음 제거 함 |
|  |  |

5.한계 및 연구방향

우선 필자가 제안한 알고리즘에는 몇 가지 한계가 있다. 우선 밝기에서 임계치(threshold)를 결정하는 통계적 방법을 제안했지만, RGB와 RGBL에서는 그렇지 않았다. 처음에는 밝기에서 임계치를 구한 것과 마찬가지로 모든 픽셀의 유사도를 구한 다음, 통계적 유의수준이 5%인 p-value를 선정하고자 했다. 하지만 이는 계산량이 매우 많다는 문제점을 가지고 있어서 구현하지 못했으며, 샘플링으로 일부 데이터를 선정하고 임계치를 결정하는 방법도 샘플의 편향 가능성 때문에 시도하지 않았다. 그리고 threshold를 통계적으로 결정해도 여전히 임계치를 1개 만 사용해서 발생한 문제가 나타나고 있다. 즉, threshold가 너무 높으면 디테일이 떨어지고 반면 너무 낮으면 잡음이 심한 문제가 발생하고 있다.

이를 해결하기 위해 현재는 edge를 분류할 때 해당 픽셀이 잡음인가를 확인하고 있지만, 다른 방법은 threshold를 2개 이상 사용한 결과를 적절한 방법으로 합치는 것이 있다. 합치는 대상은 전처리 방법은 동일하지만 서로 다른 threshold를 이용한 결과들도 있지만, 다른 전처리를 통해 얻은 edge 결과물들도 합칠 수 있다. 이는 시뮬레이션에서 봤듯이 각 전처리 방법의 장점을 합친다는 것이다. 그리고 RGB 또는 RGBL에서 픽셀을 분류하는 것은 연산량이 많을 수 있는데, 이를 극복하기 위해 thread를 이용하거나 알고리즘을 행렬로 수정하여 그래픽카드에서 연산을 수행하는 방법도 있다. 공통적으로 제안된 알고리즘들은 edge를 불연속적으로 얻는 경우가 있기 때문에, 잡음을 줄이면서 동시에 연속적으로 edge를 얻는 방법에 대해 고민해야 한다. 또한 차후에는 미분기하학적인 방법론을 이용하여 데이터를 보다 효과적으로 처리하고 edge를 얻고자 한다. 마지막으로 다른 군집알고리즘 통해 이미지 전처리를 수행할 예정이다.